이 학 석 사 학 위 논 문

Semiconductor Data Analysis via Multi-stage Stacked Generalization

다층 모델 쌓기를 통한 반도체 공정 데이터 분석

2020년 8월

서울대학교 자연과학대학원

통계학과

박 민 준

Abstract

Minjun Park
The Department of Statistics
The Graduate School
Seoul National University

This study examines the effect of multi-stage stacked generalization via semiconductor data analysis. Stacked generalization (or stacking) has been popular in the scene of machine learning, especially for data competition such as Kaggle. Stacking is an ensemble method that combines different machine learning algorithms to produce best result. It stacks the predictions from base learners and use them as input for high-level learner. For constructing ensemble models with multi-stage stacking, gradient boosting libraries such as XGBoost and LightGBM play important roles as higher-level learners. Gradient boosting libraries are widely used due to its efficiency, accuracy and interpretability and achieves better performance in stacking model than other algorithms. Through semiconductor data, we verify that multi-stage stacking model with gradient boosting libraries as high-level learners shows relatively good performance compared to single models or other stacking models.

Keywords: Stacked generalization, Gradient boosting, XGBoost, LightGBM, Semiconductor data analysis
Student Number: 2018-20823

Contents

List of Tables

Chapter 1

Introduction

A thesis of this study originates from Brightics Contest hosted by Samsung SDS. The main goal of the contest was to predict a defective rate using censor data observed during semiconductor production. My team won the second prize by using stacked generalization (or stacking) appropriately. At that time it was quite simple model with combining only 3 algorithms, mainly gradient boosting libraries, for just two stages. It leads to some thoughts on what if more algorithms or more than 2 stages we use. This was the start of my thesis.

In this paper, we will examine the effect of stacked generalization, especially focused on a case of multi-stage ensemble, mainly on semiconductor processing data. First, we will review related works on gradient boosting. Gradient boosting libraries such as XGBoost and LightGBM plays important roles as meta or higher-level learner in our stacking model. It is significant to know why they works better than other in stacking. Then, we will briefly go through the basic concept of stacked generalization. Lastly, we will ver-

ify the effect of multi-stage ensemble with stacking through semiconductor processing data via comparison between various stacking models.

Chapter 2

Overview

2.1 Gradient Boosting

2.1.1 Boosting

Boosting [1] is an ensemble method that combines many "weak" base learners to produce a powerful output from them. In order to do so, it successively trains base learners on perturbed data set. At each time perturbations are made by reweighting on the learning set based on its previous result. Boosting can be expressed as an additive expansion in a set of base learners $h(x;\ a)$, which is a function of multivariate input x characterized by parameter a. In other words,

$$f(x) = \sum_{m=1}^{M} \beta_m h(x;\ a_m) \tag{2.1}$$

where β's are the weights of corresponding base learners. We can choose a set of base learners that minimizes a loss function,

$$\min_{\{\beta_m, a_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m h(x;\ a_m)\right) \tag{2.2}$$

The minimization of such loss function is occasionally infeasible. In this case, especially for boosting, one can consider forward stagewise additive modeling,

$$(\beta_m, a_m) = \arg\min_{\beta, a} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta h(x_i;\ a)) \tag{2.3}$$

$$f_m(x) = f_{m-1}(x) + \beta_m h(x;\ a_m) \tag{2.4}$$

2.1.2 Gradient tree boosting

Gradient boosting [2] is a boosting algorithm with steepest descent adapted. Steepest descent is a numerical minimization method that updates solutions $\mathbf{f}$ toward the "steepest descent" direction, which is the negative gradient $-\mathbf{g}$ by a small amount ρ, i.e.

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m \tag{2.5}$$

where the current negative gradient $\mathbf{g}_m$ with loss function $\mathrm{L}(\cdot)$ is

$$\mathbf{g}_m = (g_{im}) = \left(\left[\frac{\partial \mathrm{L}(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}\right) \tag{2.6}$$

and the step length ρ_m is found via the "line search",

$$\rho_m = \arg\min_{\rho} \mathrm{L}(\mathbf{f}_{m-1} - \rho \mathbf{g}_m) \tag{2.7}$$

When adding a base learner in (2.4), we might choose one that is equivalent to the negative gradient $-g$ as the steepest descent method does. But the gradient is defined only at current data points so it can easily cause overfitting. To tackle this drawback, we can estimate $h_m(x;\ a)$ by fitting it to $-g_m$ via least square method,

$$a_m = \arg\min_{a,\beta} \sum_{i=1}^{N} [-g_m(x_i) - \beta h(x_i;\ a)]^2 \tag{2.8}$$

Based on obtained $h_m(x;\ a_m)$ which is most close to $-g_m$, we can get estimated ρ_m via line search,

$$\rho_m = \arg\min_{\rho} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \rho h(x_i;\ a_m)) \tag{2.9}$$

The whole procedure is summarized in Alg 1.

Algorithm 1: Gradient Boosting

---

1. Initialize $f_0(x) = \arg\min_\rho \sum_i^N L(y_i, \rho)$.

2. For m = 1 to M do:

   (a) $\tilde{y}_i = -\left[\frac{\partial \mathrm{L}(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)}, i = 1, ..., N$

   (b) $a_m = \arg\min_{a,\beta} \sum_{i=1}^N [\tilde{y}_i - \beta h(x_i;\ a)]^2$

   (c) $\rho_m = \arg\min_\rho \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \rho h(x_i;\ a_m))$

   (d) Update $f_m(x) = f_{m-1}(x) + \rho_m h(x;\ a_m)$

A special case of gradient boosting is that each base leaner $h(\cdot)$ is a J-terminal node regression tree $T(\cdot)$, i.e.,

$$h(x;\ \{b_j, R_j\}_1^J) = \mathrm{T}(x;\ \Theta) = \sum_{j=1}^{J} b_j \mathrm{I}(x \in R_j) \tag{2.10}$$

with a set of parameters, $\Theta = \{R_j, b_j\}_1^J$, where R_j's are terminal nodes of the tree, and b_j's are corresponding weights of each node. Then, the boosted tree model is the sum of such trees,

$$f_M(x) = \sum_{m=1}^{M} \mathrm{T}(x;\ \Theta_m) \tag{2.11}$$

The parameter can be estimated stagewisely,

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} \mathrm{L}(y_i, f_{m-1}(x_i) + \mathrm{T}(x_i, \Theta_m)) \tag{2.12}$$

The m-th tree also can be constructed by fitting it to $-g_m$,

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} [-g_{im} - \mathrm{T}(x_i, \Theta)]^2 \tag{2.13}$$

The loss function can be rewritten as follows,

$$\mathrm{L}(T_m) = \sum_{i=1}^{N} [-g_{im} - \mathrm{T}(x_i, \Theta)]^2 \tag{2.14}$$

$$= \sum_{i=1}^{N} [g_{im}^2 + 2 g_{im} \mathrm{T}(x_i, \Theta) + \mathrm{T}(x_i, \Theta)^2] \tag{2.15}$$

$$= \sum_{j=1}^{J} [\sum_{x_i \in R_{jm}} g_{im}^2 + 2 b_{jm} \sum_{x_i \in R_{jm}} g_{im} + n_{jm} b_{jm}^2] \tag{2.16}$$

$$= \sum_{j=1}^{J} [2 G_{jm} b_{jm} + n_{jm} b_{jm}^2] + \text{const.} \tag{2.17}$$

where G_{jm} and n_{jm} denote the sum of gradients and the number of instances in the node R_{jm} respectively. Then,

$$\tilde{b}_{jm} = -\text{ave}_{x_i \in R_{jm}} g_{im} \tag{2.18}$$

$$= -\frac{\text{G}_{jm}}{n_{jm}} \tag{2.19}$$

If we put $\tilde{b}_{jm}$ back into (2.17), the loss function becomes,

$$\text{L}(T_m) = -\sum_{j=1}^{J} \frac{\text{G}_{jm}^2}{n_{jm}} + \text{const.} \tag{2.20}$$

Based on this, for a single node j, the loss reduction after split is defined as,

$$\text{L}_{split} = \frac{\text{G}_{jmL}^2}{n_{jmL}} + \frac{\text{G}_{jmR}^2}{n_{jmR}} - \frac{\text{G}_{jm}^2}{n_{jm}} \tag{2.21}$$

i.e., the difference between the loss before and after split, where G_{jmL} is the sum of gradients in the left node after split and G_{jmR} the right one.

Algorithm 2: Exact greedy algorithm

input : I, instance set of current node

$G \leftarrow \sum_i g_i$

for *feature* $k = 1$ **to** m **do**

  $G_L \leftarrow 0$

  $gain \leftarrow 0$

  for j *in sorted*(I, *by* $x_j k$) **do**

    $G_L \leftarrow G_L + g_j$

    $G_R \leftarrow G - G_L$

    gain $\leftarrow \max(\text{gain}, \frac{\text{G}_L^2}{n_{jmL}} + \frac{\text{G}_R^2}{_{jmR}} - \frac{\text{G}^2}{n_{jm}})$

output: Split with max gain

A split point at each node will be a data point with minimum L_{split}. One can greedily visit all the data instances in current node and comparing L_{split} at all possible candidates, which is summarized in Alg 2. The whole gradient boosting procedure for regression trees is summarized in Alg 3.

Algorithm 3: Gradient Tree Boosting

---

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N}(y_i, \gamma)$

2. For $m = 1$ to M:

   (a) For $i = 1, 2, ..., N$ compute

   $$\tilde{y}_{im} = -\left[\frac{\partial \mathrm{L}(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}} \tag{2.22}$$

   (b) To construct the regression tree $\mathrm{T}(x_i, \Theta)$, or equivalently to get terminal nodes $R_{jm}, j = 1, 2, ..., J_m$, fit the tree to $\tilde{y}_{im}$ by LS

   (c) (Line search) for some loss function $\mathrm{L}(\cdot)$ and $j = 1, 2, ..., J_m$ compute

   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} \mathrm{L}(y_i, f_{m-1}(x_i) + \gamma) \tag{2.23}$$

   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathrm{I}(x \in R_{jm})$

3. Output: $\hat{f}(x) = f_M(x)$

2.2 XGBoost and LightGBM

2.2.1 XGBoost

XGBoost [3] is a fast, scalable tree boosting system applying machine learning algorithms under gradient boosting framework. XGBoost uses such regularized learning objective,

$$\mathrm{L}^{(m)} = \sum_{i=1}^{N} \mathrm{L}(y_i, f_{m-1}(x_i) + \mathrm{T}_m(x_i)) + \Omega(\mathrm{T}_m) \tag{2.24}$$

where $\Omega(f) = \gamma J + \frac{1}{2}\lambda w^2$, which penalizes the complexity of the tree model in order to prevent overfitting. It controls the number of tree nodes T so that the individual tree model be simple and predictive. Also it shrinks the weights w of each corresponding nodes for individual trees.

This learning objective can be easily optimized by second-order approximation. Then, the loss reduction after split is

$$\mathrm{L}_{split} = \frac{1}{2}\left[\frac{\mathrm{G}_{jmL}^2}{\mathrm{H}_{jmL}+\lambda} + \frac{\mathrm{G}_{jmR}^2}{\mathrm{H}_{jmR}+\lambda} - \frac{\mathrm{G}_{jm}^2}{\mathrm{H}_{jm}+\lambda}\right] - \gamma \tag{2.25}$$

Thus, split finding is conducted via greedily comparing L_{split} for all possible candidates. Exact greedy algorithm is powerful since it visits all the data instances for all the features, but it is also enormously demanding when the data size is large. For efficient proposal calculation, an approximate algorithm has been studied in past literatures. In a nutshell, the approximate algorithm picks split candidates based on percentiles of feature distribution. The algorithm allocates the continuous features into buckets split by those candidates, and finds best split point among proposals based on

aggregate statistics obtained from the buckets. Existing algorithms has proposed various approximation framework for distributed tree learning, such as histogram-based algorithm. XGBoost proposes a novel weighted quantile strategy. It is a quantile sketch for weighted datasets, and thus more appropriate for tree boosting since there are weights on dataset.

Beyond the novel approximate algorithm for split finding, XGBoost also proposes sparsity-aware split finding algorithm. It benefits from sparsity of data with respect to numerical computation. On top of that, it improves parallelized tree learning environment in a perspective of systemic design other than the algorithmic aspect. Out-of-core computation and cache-aware learning are those improvements which enables handling large scale problems with limited resources.

2.2.2 LightGBM

Gradient Boosting Decision Tree (GBDT) is powerful but there has been some issues such as unsatisfactory efficiency and scalability for large datasets with high feature dimensions. Such inefficiency mainly comes from split finding. Since it is required to scan all the data points and to calculate loss reduction by them for searching the best split point with maximum information gain, it has to be time consuming when especially the dataset is large and high-dimensional. LightGBM [4] handles such problems quite straightforwardly. It reduces the number of data instances and the number of features.

For the number of data instances, LightGBM proposes a novel sampling method called *Gradient-based One-side Sampling* (GOSS). There are no nat-

ural sample weights in GBDT. In that gradients plays an important role when computing information gain due to split, they are equivalent to usual sample weights in boosting. Data instances with larger gradients will contribute to loss reduction more than those with smaller gradients. In this regard, GOSS keeps those with large gradients to guarantee the accurate estimation of information gain, and conducts down sampling with the rest of data instances. To reduce the number of features, *Exclusive Feature Bundling* (EFB) is implemented in LightGBM. EFB groups features together, which hardly take nonzero values simultaneously. Then, the complexity of split finding becomes $O(\#\text{data} \times \#\text{bundle})$ from $O(\#\text{data} \times \#\text{feature})$. The EFB algorithm elaborates how to bundle and to merge features in same bundle, but will not be discussed in this paper.

2.3 Stacked Generalization

Stacked generalization (Stacking) [5] is an efficient ensemble method that combines multiple learners to reduce generalization error. Predictions generated from one or many base learners are used as inputs for a second-layer learning. It is empirically known that stacking is most efficient when the base learners used are different from each other as much as possible. Stacking is distinguished from bagging and boosting in that it actually combines different algorithms to produce a powerful result.

First, one need to choose base learners which will be used in phase 0. As mentioned above, the diversity of learners plays a key role in success of stacking, so base learners or single models should be as many and diverse as possible. Using various kinds of algorithms is a primal option. However, the number of available algorithms is usually limited. Another option could be perturbation on learning sets. Perturbation could be applied via various feature selection or bootstrap. After choosing a set of learners, train them on learning set with 5-fold cross-validation and perform prediction on test set with each trained model. In detail, for each training, the learning set is divided into 5 folds. One of 5 folds is held out and remaining 4 folds are used in training. The hold-out fold is predicted with this trained model. This repeats total 5 times, and one can finally produce whole cross-validation predictions for learning set. In phase 1, train a high-level learner using CV predictions in the previous phase as inputs. Prediction on test predictions, then, produces a final prediction for original test data. The above procedure is summarised in table 2.1.

Furthermore, multi-stage ensemble with stacked generalization is also pos-

sible. It just passes the CV and test predictions to the next stage, and repeats what is conducted in phase 0. Then, what it really matters is to decide which and how many learners should be used in each stage, and when to stop adding stages. One can think of using validation set from each stage to decide a composition of learners and the number of them via forward selection. However, additional stage will not always guarantee the enhancement of performance. Moreover, it is empirically known that increase of performance dramatically diminishes as the stages get deeper.

Phase 0 :	train base learners with 5-fold cross-validation, and perform prediction on test data
Phase 1 :	train a high level learner and perform prediction using the CV and test predictions in the previous phase

Table 2.1: A simple scheme for stacking

Chapter 3

Application

3.1 Dataset and Setup

The dataset used for experiment in this paper is a semiconductor processing data provided in Brightics Contest hosted by Samsung SDS. Defective rates of 3507 observations are given as a label. Features are some processed values observed from 86 censors during production. The detailed information for features was not available. The defective rates are ranged from 0.0002 to 17.185. About 100 observations are regarded as extreme values which is over than 3. For the contest, predicting such extreme values was important as a proposed score metric, Weighted Mean Squared Error (WMAE) gives large weights to those extreme values. However, a purpose of this paper is to examine an effect of stacked generalization in various settings. Therefore, in this paper, Mean Squared Error (MSE) and corresponding CV scores are

mainly used.

For base learners (or single models), a set of machine learning algorithms, which support Scikit-learn API for convenience of analysis, are selected as Random forest (RF), Extra tree (ET), Adaboost (Ada), K-Nearest Neighbors (KNN), Gradient boosting machine (GBM), XGBoost (XGB), and Light-GBM (LGBM). For each learners, hyperparameter tuning is conducted only once with learning set.

3.2 Comparison

Single models		
Models	CV score	MSE
RF	0.835	0.849
ET	0.778	0.765
Ada	0.843	0.845
KNN	0.793	0.791
GBM	0.716	0.722
XGB	0.723	0.737
LGBM	0.761	0.734

Table 3.1: Summary for single models

The results for single models is summarised in Table 3.1. Out of 7 models, GBM, XGB and LGBM show better performance than others both in terms with CV score and MSE. Based on this, 3 different models with stacked generalization will be compared. The first model (M1) is 2-stage ensemble

with stacked generalization, which phase 0 consists of all of 7 single models above. Then, for phase 1, XGB will perform as a meta learner. Their results will be compared. The second model (M2) is also 2-stage model but only 3 learners, GBM, XGB, and LGBM, are in phase 0. The last model (M3) is 3-stage ensemble with full 7 models in phase 0; GBM, XGB, and LGBM in phase 1. For phase 2, a meta learner will be similarly either XGB, or simple averaging whole stacks. The actual implementation was mainly conducted by using a Python package, *vecstack*, which enables stacking with various customization.

M1	phase 0	RF, ET, Ada, GBM, KNN, XGB, LGBM
	phase 1	XGB
M2	phase 0	GBM, XGB, LGBM
	phase 1	XGB
M3	phase 0	RF, ET, Ada, GBM, KNN, XGB, LGBM
	phase 1	GBM, XGB, LGBM
	phase 2	XGB or averaging

Table 3.2: The stacked generalization models with different setting

The overall comparison between 3 models (M1, M2, and M3) is summarised in Table 3.3. For M1, MSE is 0.699 which is about 0.02 points smaller than 0.722, the MSE of GBM which is the best score among single models. For M2, the result is quite disappointing compared to M1. There was an improvement but only about 0.001 compared to the MSE of single GBM. The composition of M2 was decided quite intuitively. The idea of using

Model	MSE (Average)
M1	0.699
M2	0.721
M3	0.696 (0.683)

Table 3.3: Comparison between M1, M2 and M3

only top 3 model for phase 0 comes from a simple intuition that GBM, XGB, and LGBM are top 3 single models with best scores among 7 algorithms. However, the result of M2 was directly against that intuition as shown in Table 3.3. This might be because the information used in M2 is relatively small since it only uses 3 models out of 7. Also, GBM, XGB, and LGBM are relatively similar to each other, compared to rest of single models. XGB and LGBM are both gradient boosting libraries so that might be another reason for such result. The MSE of M3 is 0.696 which is 0.003 smaller than M1. This is not as good as when the first stage is added to single models in M1. The additional stage brings dramatically diminished increment in performance in terms with MSE, from 0.02 to 0.003. This verifies that what we discussed about stacked generalization in Chapter 2. The choice of phase 1 for M3 was based on forward selection via cross-validation. The combination of GBM, XGB and LGBM gives best CV score. For M3, unlike M1 and M2, averaging was used to produce final output. i.e., we collected the test predictions from each stage, which is called *stack*, and averaged them to use the final prediction. Total 10 stacks were used. This gives surprisingly good result, which is 0.683 in terms of MSE.

Chapter 4

Conclusion

So far this study has examined the effect of stacked generalization. Whether it is large or small, there was definitely improvement of performance upon each additional stage. In doing so, gradient boosting libraries such as XGBoost and LightGBM played important roles as higher-level or meta learners. However, such enhancement is not always guaranteed as we already mentioned in Chapter 2. We verified that certain choices of learner sets, especially in which they are similar to each other, show no improvement or less. Furthermore, for 3-stage stacked generalization model, there has been clearly increases of performance up to phase 2 but in further phases it has no improvement at all. So the number of stages should be carefully decided.

Whether stacked generalization can be applied to real-world problem is still on question. Stacking is time consuming since it combines different algorithms. Time complexity increases depending on the number of algorithms or single models used. The hyperparameter tuning is even more demanding. The simplest way is to conduct it once at first time before stacking but it is

not the best choice. Ideally the tuning should be done at each stage for every single models. Still, stacked generalization is a descent ensemble method that combines existing fine algorithms and is of significance in that it provides the concept of meat learner in the scene of machine learning.

The future study will be deeper and larger multi-stage ensemble model with stacked generalization. To overcome the limited number of algorithms, perturbation on learning set such as various sets of features can be considered. Then, there could be several single models for each algorithm. In doing so, more validated way to determine the number of single models and stages should also be proposed.

Bibliography

[1] Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. No. 10. New York: Springer series in statistics, 2001.

[2] Friedman, Jerome H. Greedy function approximation: a gradient boosting machine. *Annals* of statistics (2001): 1189-1232.

[3] Chen, Tianqi, and Carlos Guestrin. Xgboost: A scalable tree boosting system. *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 2016.

[4] Ke, Guolin, et al. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*. 2017.

[5] Wolpert, David H. "Stacked generalization." *Neural networks* 5.2 (1992): 241-259.

국문초록

본 학위논문은 반도체 공정 데이터 분석을 통해 다층 모형 쌓기(multi-stage stacked generalization)의 효과를 살펴보는 것을 주 목적으로 한다. 모형 쌓기는 기계학습 분야, 특히 Kaggle과 같은 데이터 분석 경진대회에서 널리 사용되고 연구되었다. 서로 다른 알고리즘들을 결합하여 최적의 결과를 도출하는 앙상블 방법 중 하나이다. 다수의 학습기들을 훈련시켜 얻은 결과물을 다시 인풋으로 활용하여 더 높은 계층의 학습기로 훈련시켜 최종 결과를 도출하는 방법이다. 이런 모형 쌓기를 활용한 앙상블 모형의 구성에 있어 그라디언트 부스팅 계열의 XGBoost와 LightGBM이 높은 계층의 학습기로서 중요한 역할을 수행하였다. 그라디언트 부스팅 라이브러리들을 높은 계층의 학습기로서 활용한 다층 모형 쌓기의 효과를 단일 모형 혹은 다른 앙상블 모형들과 비교함으로써 확인하였다.

주 요 어 : 다층 모형 쌓기, XGboost, LightGBM, 반도체 공정 데이터 분석
학 번 : 2018-20823